**ENSC 180 – Introduction to Engineering Analysis Tools**

**Assignment #2: The Hound of the Baskervilles**

## Background

Statistical analysis of text is an important part of computational linguistics. Early concepts of natural language modelling date back to the work of Claude Shannon [[1]](#footnote-1)[1], who posed the question: given a sequence of letters, what is the likelihood of the next letter? To answer this question, we analyze a piece of English text for continuous sequences of *N* letters, known as   
*N*-gram statistics.

In the simplest case, *N* = 1 or “unigram”, the occurrence of each individual letter in a dataset is counted. This histogram count of letters allows us to estimate the probability of their occurrence, but is unfortunately limited since certain letter combinations, e.g. *qa* and *zbx*, cannot occur in traditional English. Hence, if we were to randomly select letters one-by-one based on our estimated probabilities, this would result in a poor approximation for the English language.

To overcome this problem, we can count longer sequences of letters. For example, *N* = 2 or “digram” statistics, counts the occurrence of *aa*, *ab*, …, *az*, *ba*, *bb*, …, etc. This allows one to estimate the likelihood of the next letter based on the previous one. For example, given the previous letter *a*, how likely is it for the next letter to be *b*? *c*? etc. This is known as a Markov model, and can be used to approximate the underlying language.

## Objectives

The assignment is intended to drill the basics of MATLAB and will be based around the following topics:

1. String processing.
2. File I/O involving text.
3. Plotting and visualization.
4. Probability and Markov models (No prior knowledge of this expected or even necessary to complete this assignment. However, some insight will prove useful; see the Appendix).

You are required to open a text file in MATLAB, read and parse its contents, and count the occurrences of certain symbols or sequences of symbols within. The histogram counts of these symbol sequences will then be used as inputs to black-box functions provided in this assignment to approximate the English language. The resulting simulations will be outputted to a single text file.

## Procedure

**Preamble:** When working with regular expressions, do not assume that your expressions will work immediately, without flaw. Your regular expression may yield unexpected results, regardless of how infallible you believe it to be. It is wise to test a regular expression against a variety of short strings in the MATLAB terminal. Only place the regular expression in your MATLAB script when it functions as intended in all cases. The same can be said, to a lesser extent, of the sprintf/fprintf functions.

Another common issue for the uninitiated is putting apostrophes within a string. Since ' is a part of MATLAB string creation syntax, we obtain an apostrophe in a string by typing it twice:

>> str = 'don''t'  
str =  
don't

Creating a string composed of a single apostrophe looks even more confusing:

>> str = ''''  
str =  
'

**Task 1:** Write a MATLAB function with the following description:

function counts = letterStatistics(filename, allowedChar, N)

This function will open a text file specified by filename and read its entire contents. The contents will be parsed such that any character that isn’t in allowedChar is removed. Finally it will return a count of all N-symbol combinations in the parsed text. This function should be stored in a file name “letterStatistics.m” and should be organized as follows:

1. Begin the function by setting the default value of N to 1 in case:
   1. The user specifies a 0 or negative value of N.
   2. The user doesn’t pass the argument N into the function, i.e.,   
      counts = letterStatistics(filename, allowedChar)
2. Using the fopen function, open the file filename for reading in text mode.
3. Using the function fscanf, read in all the contents of the opened file into a string variable.
4. There exists a MATLAB function to turn all letters in a string to lower case. Since our analysis will disregard case, use this function on the string of text.
5. Parse this string variable as follows (use logical indexing or regular expressions – do not use for loops):
   1. We want to remove all newline characters without this occurring:  
        
      Mr. Sherlock Holmes, who was usually

very late in the mornings, save

upon those not infrequent occasions

when he was up all night, was seated

at the breakfast table.

Mr. Sherlock Holmes, who was usuallyvery late in the mornings, saveupon those not infrequent occasionswhen he was up all night, was seatedat the breakfast table.   
  
 Replace all newline characters (special character \n) with a single space: ' '.

* 1. We will treat hyphenated words as two separate words, hence do the same for hyphens '-'.
  2. Remove any character that is not in allowedChar. Hint: use regexprep with an empty string '' as an argument for replace.
  3. Any sequence of two or more blank spaces should be replaced by a single blank space.

1. Use the provided PermsRep function, to create a matrix of all possible N-symbol combinations of the symbols in allowedChar.
2. Using the strfind function, count all the N-symbol combinations in the parsed text into an array counts. Do not loop through each character in your parsed text as you would in a C program.
3. Close the opened file using fclose.

**Task 2:** Write a MATLAB function with the following description:

writeString(filename, str, wordsPerLine)

This function will write a string str to an output text file filename. It will format the output such that only wordsPerLine number of words are written to each line of the text file. This function should be stored in a file name “writeString.m” and should be organized as follows:

1. Begin the function by setting the default value of wordsPerLine to 15 in case:
   1. The user specifies a 0 or negative value of wordsPerLine.
   2. The user doesn’t pass the argument wordsPerLine into the function.
2. Using the fopen function, open the file filename for appending in text mode.
3. We will define a “word” as follows: a consecutive sequence containing any number of alphabetic characters or apostrophes that is followed by 0 or more blank spaces.  
   (0, because the very last word in the string will not be followed by a blank space). Using the sprintf function, create a regular expression to match at least 1 but no more than wordsPerLine consecutive “words”.
4. Use the regexp function on str with the regular expression from the previous step. Save the starting and ending indices of all matches.
5. Iterate through all the matches and write them to a text file using the fprintf function.
6. Since we are appending data, use fprintf to output 2 newline characters afterwards. This will produce a cleaner-looking output in case of multiple calls to this function with the same filename.
7. Close the output file using fclose.

**Task 3:** Write a script to make it easier to execute these functions properly, and to create plots and visualizations. Title the script “letterStatisticsDemo.m” and organize it as follows:

1. It is wise to start any script with:  
   clear all  
   close all  
   to ensure that it is entirely self-contained and can run with an initially empty workspace.
2. In our analysis, we only wish to deal with the following characters: blank spaces, apostrophes, and letters “a” to “z”. Create an array called allowedChar containing exactly one of each aforementioned character. Calling unique(allowedChar) in the MATLAB terminal should output the same array as allowedChar.
3. Call the function letterStatistics using allowedChar from the previous step, N = 1, and 'baskervilles.txt'. Store the output in a variable counts.
4. Using the bar command, create a histogram plot of the letter occurrences in counts. Change the 'XTick' and 'XTickLabel' attributes such that each bar is labelled according to the characters in allowedChar in precisely that order. At this point, quickly check your results. The blank space character should occur most frequently. Vowels *a*, *e*, *i*, and *o* should also occur often, whereas letters like *j*, *q*, *x*, and *z* should not.
5. Use the provided function simulateIndep to create a natural language approximation according to the statistics in counts. To see how this function is called, type help simulateIndep in the MATLAB terminal. For more information, see Appendix 1.
6. Write the output of simulateIndep into a string file by calling the writeString function with the output file 'output.txt'.
7. Repeat Step 3 with N = 2.
8. In this case the vector counts contains 2-symbol occurrences (Fig. 1). We want to turn it into a matrix as shown in (Fig. 2).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | aa | 1 | | ab | 372 | | ac | 559 | | ... |  | | ba | 306 | | bb | 23 | | ... |  |   Figure 1 | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | a | b | c | ... | | a | 1 | 372 | 559 |  | | b | 306 | 23 |  |  | | c |  |  |  |  | | ... |  |  |  |  |   Figure 2 |

Do this using the reshape function and store the output in a matrix MM. Afterwards, normalize each row of MM such that sum(MM, 2) outputs a vector of 1.0000’s (perform this check in the terminal). The resulting matrix is what’s known as a Markov matrix. It contains transitional probabilities and can be interpreted as follows: the row denotes the previous letter and the column denotes the next letter, hence the value in row A and column B is the probability that the next letter will be a B, given that the previous letter was an A (see Appendix 2 for a simpler example).

1. Visualize the Markov matrix MM as an image using the imagesc function and the colormap gray. Change the 'XTick', 'XTickLabel', 'YTick', and 'YTickLabel' attributes to label each row and column of the matrix as in Fig. 2. Move the x-axis to the top using set(gca, 'XAxisLocation', 'top'). At this point, quickly confirm your results. Columns with vowels should be relatively bright in consonant rows, indicating that vowels are likely to follow consonants. The blank space character should also be likely given that the previous letter was a consonant. Finally, as a definitive check, if the previous letter was a *q*, the letter that follows can only be a *u*, therefore column *u* of row *q* should be very bright, while all other columns in that row should appear dark.
2. Use the provided function simulateMarkov to create a natural language approximation according to the 2nd order statistics in counts. To see how this function is called, type help simulateMarkov in the MATLAB terminal. For more information, see Appendix 2.
3. Write the output of simulateMarkov into a string file by calling the writeString function with the output file 'output.txt'.
4. Repeat Step 3 with N = 3.
5. Use the provided function simulateMarkov to create a natural language approximation according to the 3nd order statistics in counts.
6. Write the output of simulateMarkov into a string file by calling the writeString function with the output file 'output.txt'.
7. Repeat Steps 12 – 14 with N = 4.

## Analysis

Examine output.txt and compare the approximations for *N* = 1, 2, 3, and 4. Briefly comment on your observations.

## Appendix 1 – Probability Distributions

Let’s say you are given a coin and are asked to flip it *N* = 100 times. Each flip is independent, i.e., the outcome of the current flip is not affected by the outcome of the previous flip. The results are as follows:

THHTTHTHTTHTHTHTHHHTHTHTTHTTHHTHTTHTHTHTTHHTTHHTTHTTTTHTHTTHTTHTTHHHTTTTHHTTTHTHHHTTTTHTHHHTTTHHTHTH

If we count the occurrence of heads and tails we obtain:

H = 45  
T = 55

Based on this knowledge we can estimate the probability of the coin landing as “heads”, P(H), and “tails”, P(T), as follows:

Note that P(H) + P(T) = 1, since the coin must land either on “heads” or on “tails”; there is no other possible outcome. Based on this evidence, the possible outcomes are distributed as follows:

D:\SFU\Work & Co-op\ENSC180 - Course development\Assignments\Basics\English_Letter_Statistics\simple\coin.eps

According to our results, the coin does not appear to be fair. A biased opinion of coins is that P(H) = P(T) = 0.5, i.e., “heads” and “tails” are equally likely. Given our data, we cannot make such a claim without further evidence, i.e., more flips. In the context of this assignment, an entire novel was given for analysis rather than a few paragraphs for this very reason.

This coin can be simulated given the observed occurrences of “heads” and “tails” with the following Monte Carlo simulation in MATLAB:

mu = 0.45; %probability of heads

N = 100; %number of trials

rr = rand(1, N); %random numbers to simulate each trial

outcome = zeros(1, N);

display = 'HT'; %"H" is denoted by 1, "T" is denoted by 2

for ii = 1:length(rr)

if rr(ii) <= mu

%The current trial resulted in heads

outcome(ii) = 1;

else

%The current trial resulted in tails

outcome(ii) = 2;

end

end

display(outcome)

**Output:**

ans =

HHTTTHTTHHTTTHHHTHHHTTTTTTHTTTTTTHHTHTTHHHTTTHTTTTTTTTHTTTHHHHHHHTTTTHTHHTTHTTHHTTTHTTHHTTHHHHHTHHHT

>> heads = sum(outcome == 1)

heads =

44

>> tails = sum(outcome == 2)

tails =

56

## Appendix 2 – Markov Models

Let’s assume that the weather on any given day can be categorized as either sunny or rainy. We track the weather in Vancouver over 100 days:

RRRSRRRRRRRRRSSSSSRRRRRRRRRRRRRRRRRRRRRRRRRRSSSSRRRRRRRRRRRRRSSRRRRRRRRRRRRRRRRRRSSSRSSSRRRRRRSSRSSS

We also assume that the weather on the previous day has an effect on tomorrow’s weather (this is unlike the independent coin flips in the example in Appendix 1). Thus, we count the joint occurrences of the weather.

SS = 15  
SR = 7  
RS = 8  
RR = 69

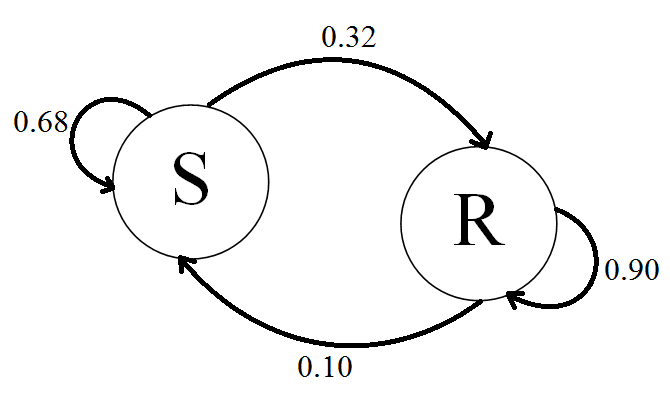
Re-organizing this data:

|  |  |  |
| --- | --- | --- |
|  | **S** | **R** |
| **S** | 15 | 7 |
| **R** | 8 | 69 |

Normalizing each row such that each row sums to 1:

|  |  |  |
| --- | --- | --- |
|  | **S** | **R** |
| **S** | 0.68 | 0.32 |
| **R** | 0.10 | 0.90 |

This Markov matrix tells us the following: If today is sunny, there is a 0.68 probability that it will be sunny again tomorrow and a 0.32 probability that it will be raining. However, if today is rainy, there is a 0.90 probability that it will continue to rain tomorrow and only a 0.10 probability that it will become sunny. In other words, if it’s raining today, it will probably rain tomorrow. This can be visualized as a state diagram:



This simple two-state Markov chain can be simulated with the following MATLAB code:

s = 0.68;

r = 0.9;

N = 100; %number of trials

rr = rand(1, N); %random numbers to simulate each trial

outcome = zeros(1, N);

display = 'SR'; %"sunny" is denoted by 1, "rainy" is denoted by 2

%Select a random initial state

outcome(1) = randi(2,1);

for ii = 2:N

if outcome(ii-1) == 1

%If today was sunny

if rr(ii) <= s

%Tomorrow will be sunny

outcome(ii) = 1;

else

%Tomorrow will be rainy

outcome(ii) = 2;

end

elseif outcome(ii-1) == 2

%If today was rainy

if rr(ii) <= r

%Tomorrow will be rainy

outcome(ii) = 2;

else

%Tomorrow will be sunny

outcome(ii) = 1;

end

end

end

display(outcome)

1. [1] C. E. Shannon, “A mathematical theory of communication”, *Bell System Technical Journal*, vol. 27. pp. 379-423, 623-656, 1948. [↑](#footnote-ref-1)